The VLDB Journal (2010) 19:45–66
DOI 10.1007/s00778-009-0168-8

SPECIAL ISSUE PAPER

Accuracy estimate and optimization techniques for SimRank computation

Dmitry Lizorkin · Pavel Velikhov · Maxim Grinev · Denis Turdakov

Received: 12 January 2009 / Revised: 16 July 2009 / Accepted: 31 August 2009 / Published online: 6 October 2009

Abstract The measure of similarity between objects is a very useful tool in many areas of computer science, including information retrieval. SimRank is a simple and intuitive measure of this kind, based on a graph-theoretic model. SimRank is typically computed iteratively, in the spirit of PageRank. However, existing work on SimRank lacks accuracy estimation of iterative computation and has discouraging time complexity. In this paper, we present a technique to estimate the accuracy of computing SimRank iteratively. This technique provides a way to find out the number of iterations required to achieve a desired accuracy when computing SimRank. We also present optimization techniques that improve the computational complexity of the iterative algorithm from $O(n^4)$ in the worst case to $\min(O(nl), O(n^3/\log_2 n))$, with n denoting the number of objects, and l denoting the number object-to-object relationships. We also introduce a threshold sieving heuristic and its accuracy estimation that further improves the efficiency of the method. As a practical illustration of our techniques, we computed SimRank scores on a subset of English Wikipedia corpus, consisting of the complete set of articles and category links.

D. Lizorkin (✉) · P. Velikhov · M. Grinev · D. Turdakov
Institute for System Programming of the Russian Academy of Sciences, B. Kommunisticheskaya avenue, 25, 109004 Moscow, Russia
e-mail: lizorkin@ispras.ru; dmitry.lizorkin@gmail.com

P. Velikhov
e-mail: pvelikhov@mac.com

M. Grinev
e-mail: maxim@grinev.net

D. Turdakov
e-mail: turdakov@ispras.ru

Keywords Similarity measure · Graph theory · SimRank · Algorithm · Computational complexity

1 Introduction

The requirement for measuring similarity between objects arises in many fields of computer science; examples include recommender systems, "related pages" queries of web search engines, document classification and clustering. Large amounts and fast growth of information require machine aid to humans for finding, classifying, and analyzing requested information. Such a range of challenges includes automatically detecting objects similar to a given object and ranking them in accordance with their similarity scores. While humans make judgment on object similarity intuitively, based on their previous experience, the task of systematically computing object similarity by a machine remains nontrivial. For practical applicability, an effective similarity measure should both reflect human intuition on objects similarity and provide reasonable computational complexity.

For the existing similarity measures, two broad categories can be outlined: (1) content- or text-based similarity measures that treat each object as a bag of items or as a vector of word weights, and (2) link-based ones that consider object-to-object relations expressed in terms of links. In the recent research [23], the extensive evaluation of different similarity measures was performed, and link-based measures produced systematically better correlation with human judgments compared to text-based measures. It is worth mentioning that the success of the Google search engine began with its ability to rank search results in accordance with human expectations; the latter feature was essentially based on a purely link-based ranking algorithm called PageRank [4]. From this perspective, it is reasonable to assume that an effective similarity

measure would have a comparable impact for computer science techniques as PageRank ranking algorithm had for web search.

Considering the outlined state of the art, the similarity measure *SimRank* [12] can be considered as one of the promising ones, due to the following reasons. SimRank is a link-based similarity measure, and builds on the approach of previously existing link-based measures. SimRank is based on both a clear human intuition and a solid theoretical background. Similarly to PageRank, SimRank is defined recursively with respect to a "random surfer" model and is computed iteratively. Unlike the similarity measures that require human-built hierarchies, SimRank is applicable to any domain with object-to-object relationships, including the Web.

Nevertheless, existing work on SimRank lacks two important issues. First, although SimRank iterative similarity scores are known to converge [12], a real-life computation naturally involves performing a finite number of iterations. However, a potential difference between SimRank iterative similarity scores and theoretical ones remains an open question. The symmetric question is finding out the precise number of iterations sufficient to guarantee a desired accuracy.

Second, optimization issue of SimRank computation is not the primary focus of the original SimRank proposal [12]. To the best of our knowledge, only one research paper [7] is focused on optimizing the computation of SimRank. That paper is initially oriented on SimRank *probabilistic* computation, details are considered in the Related Work section. As for SimRank iterative computation, optimization has not been addressed in scientific literature yet, and the time complexity of the straightforward SimRank computation is an obstacle for using SimRank on practical data corpora.

This paper presents a solution to both issues, mathematically proven and practically justified by experimental results. In summary, the main contributions of this paper are the following:

- A precise accuracy estimate is presented for SimRank scores computed iteratively, with respect to the theoretical ones. This allows one to find out the number of iterations required for achieving the desired accuracy.
- Optimization techniques that improve SimRank computational complexity from $O(n^4)$ to $\min(O(nl), O(\frac{n^3}{\log_2 n}))$. A threshold sieving heuristic is introduced and its accuracy estimation is given that further improves the efficiency of the method.
- SimRank computational viability for relatively large object corpora in the presence of the suggested optimization techniques is verified experimentally by computing SimRank similarity scores for a subset of English Wikipedia corpus, consisting the complete set of articles and category links.

While in the conference version of our paper [21], we reported the computational complexity of our techniques be $O(n^3)$, in this extended version of the paper, we present the additional optimization technique and improve SimRank computational complexity to $\min(O(nl), O(\frac{n^3}{\log_2 n}))$. The additional technique is essentially based on algorithms for optimized matrix multiplication.

The rest of the paper is organized as follows. In the next section, SimRank overview is given and the necessary notations and formulas are introduced. In Sect. 3, accuracy estimate for the SimRank iterative computation model is established. In Sect. 4, SimRank optimization techniques are suggested, and Sect. 5 summarizes them into integral algorithms for SimRank optimized computation. Section 6 gives the overview of the related work. Experimental results are presented in Sect. 7. Future work is discussed in Sect. 8.

2 SimRank overview

In this section, SimRank overview is given, and notations, formulas and SimRank properties necessary for further discussion are provided. The material presented in this section recalls Jeh's and Widom's work [12].

SimRank approach is focused on "object-to-object relationships found in many domains of interest" [12]. From the relationships perspective, a domain is assumed to be modeled as a (logical) graph, with nodes representing objects and edges (links) representing relationships.

The *basic intuition* behind SimRank approach is: "two objects are similar if they are referenced by similar objects" [12]. Note that the given intuition is recursive by nature. As the base case, any object is considered maximally similar to itself, i.e. having a similarity score of 1 assigned.

Before presenting the mathematical formula that reifies the basic SimRank intuition, several notations are introduced. Given a graph $G(V, E)$ consisting of a set of nodes V and a set of links E, the following two mappings are further assumed defined for each node v in the graph:

- $I(v)$ denotes all in-neighbours of node v, i.e. all nodes that have a link to v:

$$I(v) = \{u \in V \mid (u, v) \in E\}.$$

- $O(v)$ denotes all out-neighbours of v, i.e. all nodes the node v has a link to:

$$O(v) = \{w \in V \mid (v, w) \in E\}.$$

Notations $|I(v)|$ and $|O(v)|$ denote the number of nodes in $I(v)$ and $O(v)$, respectively. Individual member of $I(v)$ and $O(v)$ are referred to as $I_i(v)$, $1 \le i \le |I(v)|$ and $O_i(v)$, $1 \le i \le |O(v)|$; a particular order of members when associated with indices is not important for further discussion.

With the similarity score between objects a and b denoted by $s(a, b) \in [0, 1]$, the basic SimRank intuition is then written as follows:

$$s(a, a) = 1,$$
$$s(a, b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b)), \tag{1}$$

with a constant C being the decay factor, $0 < C < 1$. For preventing division by zero in the general formula (1) in case of $I(a)$ or $I(b)$ being an empty set, $s(a, b)$ is specially defined as zero for $I(a) = \emptyset$ or $I(b) = \emptyset$.

We will further refer to $s(*, *)$ as SimRank *theoretical* similarity function, and refer to its value $s(a, b)$ as theoretical similarity score between nodes a and b.

A solution to the SimRank equations (1) is reached by iteration to a fixed-point. For each iteration k, an *iterative* similarity function $R_k(*, *)$ is introduced, with $R_k(a, b)$ denoting the iterative similarity score between a and b on iteration k. The iterative computation starts with $R_0(*, *)$ defined as

$$R_0(a, b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{if } a \ne b. \end{cases} \tag{2}$$

On the $(k + 1)$th iteration, $R_{k+1}(*, *)$ is defined in special cases as

$$R_{k+1}(a, b) = 1, \quad \text{if } a = b,$$
$$R_{k+1}(a, b) = 0, \quad \text{if } I(a) = \emptyset \text{ or } I(b) = \emptyset,$$

and is computed from $R_k(*, *)$ in the general case as follows:

$$R_{k+1}(a, b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} R_k(I_i(a), I_j(b)). \tag{3}$$

The following SimRank properties stated in [12] are worth noting for the purposes of our further discussion:

1. A solution $s(*, *)$ to SimRank equations (1) always exists and is unique, and $s(*, *) \in [0, 1]$.
2. For each k, $R_k(*, *)$ is a lower bound on the theoretical SimRank function $s(*, *)$, i.e. $R_k(a, b) \le s(a, b)$.
3. Iterative functions $R_k(*, *)$ converge to SimRank theoretical function $s(*, *)$, i.e. $\lim_{k\to\infty} R_k(a, b) = s(a, b)$.

We will further refer to these properties by their corresponding item numbers within the above list.

Symbol K further denotes the total number of iterations performed; n denotes the number of nodes in a graph.

3 Iterative similarity accuracy estimate

Although Jeh and Widom proved iterative similarity convergence [12], SimRank practical computation naturally implies performing a finite number of iterations. From this perspective, no quantitative estimates were given for a potential difference between SimRank iterative similarity scores and theoretical ones. In this section, we fill in this gap and estimate the accuracy of computing SimRank iteratively.

The following proposition establishes the accuracy estimate for iterative similarity function $R_k(*, *)$ obtained after k iterations with respect to theoretical similarity function $s(*, *)$.

Proposition 1 *The difference between SimRank theoretical and iterative similarity scores decreases exponentially in the number of iterations and uniformly for every pair of nodes. Precisely, for every iteration number $k = 0, 1, 2, \ldots$ and for every two nodes a, b, the following estimate holds:*

$$s(a, b) - R_k(a, b) \le C^{k+1}. \tag{4}$$

In conjunction with SimRank Property 2 listed above, the proposition gives the following estimate:

$$0 \le s(a, b) - R_k(a, b) \le C^{k+1}.$$

The proof of the proposition is given in Appendix A. As an accompanying result, it follows from the proposition that $R_k(*, *)$ converges to $s(*, *)$ uniformly. Moreover, it can also be noted that the established upper bound is precise:

Notes 1 The upper bound stated in Proposition 1 is precise.

The example that verifies the Note is given in Appendix B.

Proposition 1 shows that the number of iterations K required for achieving a desired accuracy depends on neither the number of nodes in an input graph nor on any other graph characteristics like the degree of nodes. Jeh and Widom observed K independence from an input graph experimentally [12]; Proposition 1 now gives the theoretical foundation for this observation. The observation has an important implication for SimRank computational complexity, as the latter naturally depends on the number of iterations. In accordance with Proposition 1, the number of iterations can now be considered constant with respect to different input graphs.

For comparison, speed of convergence for PageRank generally depends on the structure of the modified adjacency matrix—the so-called Google matrix. For instance, PageRank scores are traditionally computed using iterative eigenvector algorithms, such as the power method, or Jacobi or Gauss-Seidel methods. For these three classes of algorithms, their speed of convergence is geometric with ratios $|\lambda_1/\lambda_2|$, $-D^{-1}(L + U)$ and $(-L + D)^{-1}U$, respectively, where λ_1 and λ_2 are the first and the second dominant eigenvalues,

D, L and U are the diagonal, the lower and the upper triangular matrices of the LDU decomposition of the Google matrix. Although it is known that these PageRank algorithms converge for the Google matrix, we have not heard of any estimations for the speed of convergence that are independent of the matrix structure.

It is worth noting that Jeh and Widom [12] suggested choosing the decay factor value $C = 0.8$ and the total number of iterations $K = 5$, which in accordance with Proposition 1 stated above imply a relatively large potential difference between theoretical and computed similarity scores:

$$0.8^{5+1} = 0.8^6 > 0.26,$$

that could be unacceptable for many domains recalling that similarity scores fall into the [0..1] segment. For guaranteeing more accurate computation results, it can be advised using either a smaller decay factor or more iterations. In [7], it was experimentally observed that a smaller decay factor indeed provides similarity scores of better quality, the observation now explained theoretically.

Depending on the needs of a particular application, Proposition 1 provides a precise mechanism for finding out (1) either the accuracy value for the given decay factor C and the number of iterations k, or (2) vice versa, the number of iterations required for achieving a desired accuracy. A more detailed discussion on the subject is given in Subsect. 4.4 below.

Proposition 1 provides an estimate for *absolute* difference between theoretical and iterative similarity scores; for ranking purposes, the relative order of nodes with respect to their similarity to a given node is generally more important than absolute scores. From this perspective, the following ranking accuracy estimate can be established.

Proposition 2 *The minimum difference between theoretical similarity scores that allows correctly ranking two nodes with respect to a third node in accordance with their pairwise iterative similarity scores decreases exponentially in the number of iterations. Precisely, if*

$$s(a,b) > s(a,d) + C^{k+1}, \tag{5}$$

then it necessarily follows that

$$R_k(a,b) > R_k(a,d). \tag{6}$$

Proof Let us estimate the difference:

$$\begin{aligned} R_k(a,b) - R_k(a,d) &> \{\text{using (5)}\} \\ &> R_k(a,b) - R_k(a,d) - s(a,b) + s(a,d) + C^{k+1} \\ &= \underbrace{s(a,d) - R_k(a,d)}_{\geq 0} - \underbrace{(s(a,b) - R_k(a,b))}_{\leq C^{k+1} \text{ using (4)}} + C^{k+1} \\ &\geq 0 - C^{k+1} + C^{k+1} = 0, \end{aligned}$$

which gives (6). □

The proposition states that the iterative SimRank computation appropriately grabs exponentially smaller differences in theoretical similarity scores.

Propositions 1 and 2 are important not only on their own, establishing an a priori correlation between the number of iterations k and iterative similarity scores accuracy, but also as a theoretical basis for one of the optimization techniques suggested in the next section.

4 Optimization techniques

In this section, optimization techniques for SimRank computation are suggested. The optimization techniques cover three consecutive aspects in SimRank computation. First, a technique for selecting essential node pairs is presented, aimed at skipping node pairs that do not require similarity scores computing for a given iteration. Second, two techniques are suggested for reducing the number of access operations to the iterative similarity function and for facilitating efficient clustering. Third, threshold-sieved similarities are introduced for speeding up subsequent iterations.

Each of the proposed optimization techniques is covered in its own subsection accordingly. In the next section, the optimization techniques are integrated into a general SimRank optimized computation algorithm.

When discussing the computational complexity provided by each optimization technique, n denotes the number of nodes in the graph, l denotes the number of links in the graph. Since each link (u, v) contributes to exactly one member of the sets $I(v)$ and $O(u)$, the total number of links correlates with cardinalities of $I(v)$ and $O(u)$ as follows:[1]

$$l = \sum_{v \in V} |I(v)| = \sum_{u \in V} |O(u)|. \tag{7}$$

4.1 Selecting essential node pairs

In this subsection, techniques for selecting essential node pairs are suggested that allow reducing the computation of iterative similarity scores to these node pairs only.

Definition 1 Let *essential paired nodes* for a given node a over a similarity function $R_k(*,*)$ denote all nodes from the following set of nodes:

$$\begin{aligned} &\text{Essential}_{R_k}(a) \\ &= \{b \mid \exists u \in I(a), \exists v : R_k(u,v) \neq 0,\ \ b \in O(v)\}. \end{aligned} \tag{8}$$

The reason for referring to nodes from a thus defined set as "essential paired nodes" for a given node becomes clear from the following proposition.

[1] Recall that notation $O(u)$ denotes the set of all out-neighbour nodes of node u, as mentioned in Sect. 2.

Proposition 3 *For a given node a, $R_{k+1}(a,b)$ is zero for every node b such that*

$$b \notin \{a\} \cup Essential_{R_k}(a). \tag{9}$$

Proof (by contradiction) Suppose that $R_{k+1}(a,b)$ is non-zero for some node b from (9). Since it follows from (9) that $b \neq a$, and since $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$ in accordance with our supposition $R_{k+1}(a,b) \neq 0$, then $R_{k+1}(a,b)$ is computed by general iterative formula (3). As all items in (3) are non-negative, it follows from $R_{k+1}(a,b) \neq 0$ that there exists at least a single pair of indexes (i_0, j_0) such that $R_k(I_{i_0}(a), I_{j_0}(b)) \neq 0$. Let us denote $I_{i_0}(a) = u, I_{j_0}(b) = v$.

By their notation, $u \in I(a), v \in I(b)$. By symmetry, it follows from $v \in I(b)$ that $b \in O(v)$. Thus $b \in \text{Essential}_{R_k}(a)$, since the chosen nodes u and v are the ones that satisfy (8). The latter result leads to a contradiction with the initial terms of the proposition. □

Definition 1 and Proposition 3 provide an algorithm for considering only essential node pairs and thus skipping iterative scores computation for the remaining ones. From the computational viewpoint, the set of essential paired nodes $\text{Essential}_{R_k}(a)$ for a given node a can be obtained by first constructing a temporary set of nodes $\text{Temp}_{R_k}(a)$ that consists of all nodes having non-zero similarity scores with some $I(a)$ member:

$$\text{Temp}_{R_k}(a) = \{v \mid \exists u \in I(a) : \ R_k(u,v) \neq 0\}. \tag{10}$$

It can easily be verified that the set of essential paired nodes for a can then be obtained by taking all out-neighbours for every node in $\text{Temp}_{R_k}(a)$, i.e.:

$$\text{Essential}_{R_k}(a) = \{b \mid \exists v \in \text{Temp}_{R_k}(a) : b \in O(v)\}. \tag{11}$$

Considering the computational complexity of obtaining essential paired nodes for a node a by formulas (10) and (11), the following observation can be drawn:

1. Calculating $\text{Temp}_{R_k}(a)$ implies scanning over each member $u \in I(a)$ and retrieving all nodes v that have non-zero iterative similarity with u, i.e. at most n nodes for each u. These calculations require performing at most $|I(a)| \cdot n$ operations.
2. Calculating $\text{Essential}_{R_k}(a)$ by formula (11) implies making a union of sets $O(v)$ over all $v \in \text{Temp}_{R_k}(a)$. Since $\text{Temp}_{R_k}(a)$ is a subset of V, the upper bound for the required number of operations is:

$$\sum_{v \in \text{Temp}_{R_k}(a)} |O(v)| \leq \sum_{v \in V} |O(v)| = l.$$

Intermediate memory consumption for both $\text{Temp}_{R_k}(a)$ and $\text{Essential}_{R_k}(a)$ is linear.

For a complete iteration, the computational complexity of selecting essential node pairs is thus a sum of the above numbers of operations over all $a \in V$:

$$\sum_{a \in V} (|I(a)| \cdot n + l) = n \underbrace{\sum_{a \in V} |I(a)|}_{l} + l \underbrace{\sum_{a \in V} 1}_{n} = 2nl.$$

Memory consumption remains linear, since $\text{Temp}_{R_k}(a)$ can be freed after $\text{Essential}_{R_k}(a)$ is constructed, and $\text{Essential}_{R_k}(a)$ can be freed after essential paired nodes for node a are processed. Further discussion in this section shows that the computational complexity is no more than $O(nl)$ for subsequent processing within a SimRank iteration as well.

Essential node pairs provide a better selectivity when the iterative similarity function $R_k(*,*)$ has a relatively small fraction of non-zero values with respect to zero ones. We will refer to such a similarity function as a *sparse* one. A technique for keeping an intermediate iterative similarity function sparse is suggested in Subsect. 4.4.

Iterative similarity scores computation can be skipped not only for node pairs with a priori zero scores, but also for the ones that are not required for a subsequent iterative computation.

Proposition 4 *If a node u in a graph has no outgoing links, then $R_{k+1}(*,*)$ does not depend on $R_k(u,*)$.*

Proof (by contradiction) Suppose that there exists a pair of nodes $a, b \in V$ such that $R_{k+1}(a,b)$ depends on $R_k(u,*)$. Note that $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$, since in a otherwise case $R_{k+1}(a,b)$ is constant by definition and thus does not depend on $R_k(u,*)$.

It follows from $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$ that $R_{k+1}(a,b)$ is calculated by the general iterative formula (3). Since $R_k(*,*)$ is presented in the right-hand side of (3) in the form of $R_k(I_i(a), I_j(b))$, the only way for $R_{k+1}(a,b)$ to depend on $R_k(u,*)$ is to have $u = I_{i_0}(a)$ for some index i_0, i.e. to have $u \in I(a)$. But $u \in I(a)$ implies by symmetry that $a \in O(u)$. The latter contradicts the proposition terms on $O(u) = \emptyset$. □

Corollary 1 *If a node u in a graph has no outgoing links, then it is sufficient to calculate $R_k(u,*)$ on just the last iteration without violating the semantics of SimRank iterative computation.*

An a priori knowledge of the precise number of iterations provided by Proposition 1 plays the crucial role for the practical applicability of Corollary 1. If the number of iterations were unknown a priori, all non-zero similarity scores would have had to be computed anyway for the reason of finding out when to terminate the iterative computation.

From the computational viewpoint, checking the applicability of Corollary 1 for a given node u can be performed in

constant time and requires no additional memory, and thus constitutes a practical pruning mechanism.

While Proposition 3 is focused on pruning node pairs with zero similarity from consideration, Proposition 4 and Corollary 1 cover a different pruning aspect. Indeed, even if a node u has no outgoing links, it can still have many incoming links and thus have a non-zero similarity with many other nodes in a graph. Moreover, calculating all these similarity scores can involve a large computational effort. However, if the node u has no outgoing links, then this computational effort can be saved on intermediate iterations and performed for the last iteration only. We will show the practical importance of this proposition when considering the experimental results.

4.2 Partial sums

After essential node pairs for a given node are selected, the optimization technique presented in this subsection allows reducing the number of access operations to $R_k(*,*)$ required for computing $R_{k+1}(*,*)$. The main idea behind the optimization is that a sum of $R_k(*,*)$ values over a certain set of arguments is used for computing *several* values of $R_{k+1}(*,*)$ and can thus be effectively memoized [1] for preventing repeated computation.

Figure 1 depicts the basic idea behind the technique. Suppose that each square in the figure represents a value of the iterative similarity function, with the first argument of the function represented by the row the square is located in, and the second argument—by the column. Then, computing similarity score for a node pair (a, b) in accordance with SimRank iterative formula (3) implies summing up the values that are shown in Fig. 1 in squares filled with horizontal lines. In the same way, squares filled with vertical lines show the values that are summed up for computing similarity score for a node pair (a, d). It is clearly visible from the figure that if $I(b) \cap I(d) \neq \emptyset$, then certain values are summed up in this example twice (namely, the ones shown in squares filled with both horizontal and vertical lines). To save the computational effort, we can sum up the values over the rows in $I(a)$ only once for each column, and then use these pre-computed partial sums for obtaining similarity scores of the node a with *all the other* nodes in the graph.

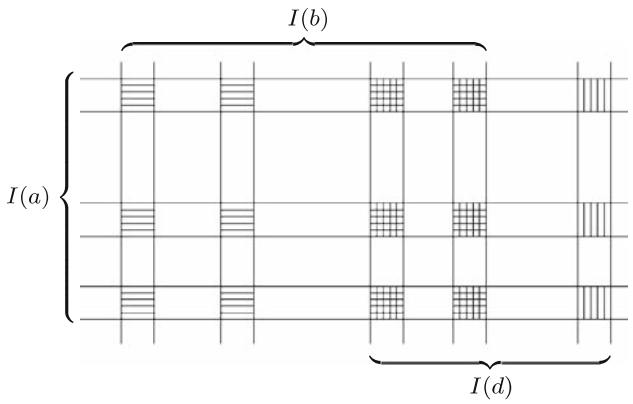

Fig. 1 Similarity scores involved in computing similarity of node pairs (a, b) and (a, d)

For an elaborate discussion on the subject, let us first introduce the notion of a *partial sums function*.

Definition 2 Let $f(*,*)$ be a binary function $X \times Y \to \mathbb{R}$ and let S be a finite subset in X: $S = \{x_1, x_2, \ldots, x_p\}$, $x_i \in X, i \in \overline{1, p}$. By partial sums for the function f over the set S we will call a unary function $Y \to \mathbb{R}$ denoted as $\text{Partial}_S^f(*)$ and defined as follows:

$$\text{Partial}_S^f(y) = \sum_{x_i \in S} f(x_i, y), \quad y \in Y.$$

A partial sums function is introduced for being applied to SimRank iterative similarity scores computation:

Proposition 5 *For $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$, $R_{k+1}(a, b)$ can be computed iteratively as*

$$R_{k+1}(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{j=1}^{|I(b)|} Partial_{I(a)}^{R_k}(I_j(b)). \qquad (12)$$

Proof The proposition is proved by simply swapping the summation signs in the iterative formula (3):

$$R_{k+1}(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{j=1}^{|I(b)|} \underbrace{\sum_{i=1}^{|I(a)|} R_k(I_i(a), I_j(b))}_{\text{Partial}_{I(a)}^{R_k}(I_j(b))}$$

and by noting that the internal summation is the value of the partial sums function for $R_k(*,*)$ over $I(a)$ for argument $I_j(b)$. □

Although trivial in its proof, Proposition 5 provides the efficient speedup technique for SimRank computation, based on the following corollary:

Corollary 2 *For a given fixed node a, the same partial sums function $Partial_{I(a)}^{R_k}(*)$ is used for computing $R_{k+1}(a, b)$ for every node b in a graph.*

The key point in optimizing $R_{k+1}(a,*)$ computation via a partial sums function $\text{Partial}_{I(a)}^{R_k}(*)$ is that once calculated, partial sums values are memoized and are thus not recalculated when subsequently required. For example, if $I_j(b) = I_l(d) = u$ for some nodes b and d, then the partial sum value $\text{Partial}_{I(a)}^{R_k}(u)$ is calculated once and is used in both $R_{k+1}(a, b)$ and $R_{k+1}(a, d)$ computation. Since for some nodes w, the partial sum values $\text{Partial}_{I(a)}^{R_k}(w)$ will probably not be required for computing the $R_{k+1}(a,*)$ values, it is reasonable to calculate the partial sums function in a delayed fashion.

Analyzing the computational complexity, the straightforward iterative SimRank computation involves $|I(a)| \cdot |I(b)|$ access operations to $R_k(*,*)$ for computing $R_{k+1}(a,b)$ for a single pair of nodes (a,b), resulting in $n^2 \text{avg}_{a,b}(|I(a)|\,|I(b)|)$ operations per iteration [12], the latter being $O(n^4)$ in the worst case [31]. For comparison, the computational complexity of performing an iteration using partial sums is obtained by considering the following two components:

1. Calculating a single value of a partial sums function $\text{Partial}^{R_k}_{I(a)}(y)$ involves summing up $|I(a)|$ values, resulting in at most $|I(a)| \cdot n$ operations for calculating all the required values of the partial sums function.
2. With the values of the partial sums function calculated, computing similarity between the node a and an arbitrary node b implies taking the sum $\sum_{y \in I(b)} \text{Partial}^{R_k}_{I(a)}(y)$, resulting in $|I(b)|$ operations.

The computational complexity of performing a complete iteration is thus expressed as the total number of operations for the above two components for every node a in the graph, that is:

$$\sum_{a \in V} (|I(a)| \cdot n + l) = n \sum_{a \in V} |I(a)| + ln = 2nl.$$

Note that both partial sums and the selection of essential node pairs exhibit the same computational complexity, which makes it attractive use the combination of both techniques when performing an iteration.

Note that although the number of operations in case of $l < n$ can be fewer than the total number of pair-wise node combinations, there is no inconsistency here. Indeed, in this case, there are at most l nodes with incoming links in a graph, that leaves only l^2 similarity scores to be computed at most.

Partial sums allow additionally speeding up SimRank computation by $R_k(*,*)$ values clustering. Precisely, before partial sums were introduced, computing $R_{k+1}(a,b)$ for a single pair of argument nodes generally required accessing $R_k(*,*)$ for a Cartesian product of $I(a) \times I(b)$; such argument values spread hardly made any clustering strategy provide access time speed up. For comparison, partial sums usage allows clustering the underlying storage for $R_k(*,*)$ values by the first argument. Indeed, due to $R_k(*,*)$ symmetry, $\text{Partial}^{R_k}_{I(a)}(u)$ can be computed from $R_k(u,*)$ involving a single node for the first argument:

$$\text{Partial}^{R_k}_{I(a)}(u) = \sum_{i=1}^{|I(a)|} R_k(I_i(a), u) = \sum_{i=1}^{|I(a)|} R_k(u, I_i(a)).$$

The above made observations allow achieving access operations speed up by clustering $R_k(*,*)$ by the first argument from the underlying storage viewpoint.

4.3 Cross summation

The idea of partial sums introduced in the previous subsection can be further generalized by grouping operations over an iterative similarity function with common arguments together. The technique presented in this subsection is essentially based on the existing algorithms for optimized matrix multiplication [15,17], specialized for the SimRank model.

Definition 3 Let $f(*,*)$ be a binary function $X \times Y \to \mathbb{R}$ and let $G(*)$ be a mapping $X \to 2^X$ from X to a finite set in X. By outer sum for the function f over the function G, we refer to a binary function $X \times Y \to \mathbb{R}$ that is defined as follows:

$$\text{Outer}^f_G(x,y) = \sum_{x_i \in G(x)} f(x_i, y), \quad x \in X, \quad y \in Y. \tag{13}$$

Proposition 6 *In the general case of $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$, the iterative similarity score $R_{k+1}(a,b)$ can be computed as:*

$$R_{k+1}(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{j=1}^{|I(b)|} Outer^{R_k}_I(a, I_j(b)). \tag{14}$$

Proof In the similar manner as for Proposition 5, let us swap the summation signs in the iterative formula (3):

$$R_{k+1}(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{j=1}^{|I(b)|} \underbrace{\sum_{x \in I(a)} R_k(x, I_j(b))}_{\text{Outer}^{R_k}_I(a, I_j(b))}.$$

Noticing that the internal summation is the value of the outer sum function for $R_k(*,*)$ over $I(a)$ for the argument pair $(a, I_j(b))$ finishes the proof. □

The key corollary of the above proposition that allows working out the optimized iterative computation is that *the single* outer sum function is used throughout a complete iteration. With this observation, it is suggested to computationally organize an iteration in two steps:

1. The first step involves computing the outer sums function $\text{Outer}^{R_k}_I(*,*)$. The step has the effect of summing up the values of the iterative similarity function $R_k(*,*)$ for *all* the combinations of the first argument appearing in the outer summation sign of the SimRank iterative formula (3). It is due to this observation that the function introduced in Definition 3 was mnemonically called the "outer sums function". We will further refer to the values of this function as simply *outer sums*.
2. On the second step, iterative similarity scores for the current iteration $R_{k+1}(*,*)$ are computed using the already computed outer sums function. The step has the effect of

summing up the outer sums over all the combination of the second argument appearing in the inner summation sign of the SimRank iterative formula (3).

We called this technique "cross summation", because summation of iterative similarity scores is separated into two distinct consecutive steps, with the first step collecting all the required summations over the first argument, and the second step further composing these outer sums over the second argument.

Using cross summation, SimRank computation can be additionally optimized by calculating multiple sums at once. For this purpose, let us choose some natural $p \leq n$ and split the whole set of nodes V of the graph into several subsets, with each subset containing at most p nodes. Formally, we introduce subsets $V_1, V_2, \ldots, V_q$, where $q = \lceil \frac{n}{p} \rceil$ and:

$$V_i \subset V, \ |V_i| \leq p, \ i = \overline{1,q};$$
$$V_i \cap V_j = \emptyset, \ i \neq j;$$
$$\bigcup_{i=\overline{1,q}} V_i = V.$$

We achieve computing the values of $\mathrm{Outer}_I^{R_k}(*, b)$ at once by summing the values of the iterative similarity function $R_k(*, b)$ for *all* possible combinations of nodes from V_j for each j. For formalizing this principle, we introduce the notion of a *combination* function:

Definition 4 For a set of nodes V_j and an iterative similarity function $R_k(*, b)$ with its second argument fixed to a node b, a unary function $\mathrm{Combination}^j_{R_k(*,b)}(*)$ is defined for each subset W_j of V_j as

$$\mathrm{Combination}^j_{R_k(*,b)}(W_j) = \sum_{a \in W_j} R_k(a, b), \quad W_j \subset V_j.$$

The function $\mathrm{Combination}^j_{R_k(*,b)}(*)$ is defined for $2^{|V_j|} \leq 2^p$ different arguments. Note further that all values of the combination function can be computed in at most 2^p summations, since each value of the function differs from some of its other values by a single summation:

$$\begin{cases} \mathrm{Combination}^j_{R_k(*,b)}(\emptyset) = 0; \\ \mathrm{Combination}^j_{R_k(*,b)}(W_j \cup u) \\ \qquad = \mathrm{Combination}^j_{R_k(*,b)}(W_j) + R_k(u, b). \end{cases}$$

With the combination function introduced, each value of the outer sums function can be computed as follows:

$$\mathrm{Outer}_I^{R_k}(a, b) = \sum_{j=1}^{q} \mathrm{Combination}^j_{R_k(*,b)}(I(a) \cap V_j).$$

Since set intersections $(I(a) \cap V_j)$ can be computed for all $a \in V$, $j = \overline{1,q}$ in advance, we introduce the notion of a *bucket* function to grab this observation:

Definition 5 For an in-neighbour mapping $I(*)$ and a sequence of sets of nodes $\{V_i\}$, $i = \overline{1,q}$, a binary function $\mathrm{Bucket}_I^{\{V_i\}}(*, *)$ is defined for its first argument being a node $a \in V$ and the second argument being an index $j \in \{1, 2, \ldots, q\}$, as follows:

$$\mathrm{Bucket}_I^{\{V_i\}}(a, j) = I(a) \cap V_j.$$

Using the bucket function, each value of the outer sums function is now computed as:

$$\mathrm{Outer}_I^{R_k}(a, b) = \sum_{j=1}^{q} \mathrm{Combination}^j_{R_k(*,b)}(\mathrm{Bucket}_I^{\{V_i\}}(a, j)). \quad (15)$$

The computational complexity for the technique is comprised of the following three components:

1. Computing the values of a single combination function requires 2^p summations, giving totally $nq2^p$ summations per a complete iteration due to nq different combination functions involved.
2. A single value of the outer sums function is computed in q summation, yielding n^2q summations per iteration.
3. The bucket function is computed in n^2 operations.

It can be observed that the number of operations for the described approach is asymptotically minimized for $p = \lfloor \log_2 n \rfloor$, with the number of operations then being $O(n^2 \frac{n}{\log_2 n}) = O(\frac{n^3}{\log_2 n})$. Comparing this computational complexity with the one for partial sums introduced in the previous subsection, it can be noted that one of the two techniques can be computationally beneficial over the other depending on the proportion between n and l. This issue is discussed in more detail below.

Inner summation is computationally organized anal ogously to outer summation and yields the same computational complexity.

Concerning space complexity, applying cross summation requires storing the values of the outer sum function, that takes quadratic space in the number of nodes in the graph. The bucket function, once computed, is used in the cross summation technique instead of the initial mapping $I(*)$, while requiring no more storage space than $I(*)$. Finally, no more that a single combination function is required at a time, involving space proportional to $2^p \leq 2^{\log_2 n} = n$, that is, linear in the number of nodes in the input graph.

Cross summation and partial sums can be considered as alternative techniques. Partial sums employ lazy evaluation, while cross summation is an eager technique. Since all outer sums are computed in advance, essential paired nodes are not applicable to cross summation; however, Corollary 1 still applies. Either partial sums or cross summation can be

preferable in different situations, more discussion on this subject is given in Sects. 5 and 7.

4.4 Threshold-sieved similarity

For certain graphs classes like scale-free graphs [16], the iterative similarity function $R_k(*,*)$ has an abundance of non-zero values after just a few iterations. However, many of these values, although being non-zero, denote low similarity between node pairs and thus contain little practical information for the result similarity scores. On the other hand, keeping all these small but nevertheless non-zero similarity scores requires considerable storage amounts and slows down subsequent iterations.

The proposition given in the following provides the quantitative illustration for the number of non-zero similarity scores through SimRank iterative computation. As a preliminary step, let us repeat the definition of the diameter of a graph, introduced in [5]:

Definition 6 The diameter of a graph is defined as the average distance between any two nodes in the graph.

If a graph contains a pair of nodes that are not connected by a (directed) path, then the diameter of the largest strongly connected component in the graph is considered. In the following, let n' denote the number of nodes in the largest strongly connected component of the graph, $n' \le n$, let d denote the diameter of that component.

Proposition 7 *If the diameter d and the number of SimRank iterations K satisfy the inequality* $2 < d \le K$*, then, denoting* $D = \lceil d \rceil$*, the lower bound for the number of non-zero values of the iterative similarity function* $R_K(*,*)$ *obtained after K iterations is:*

$$\#nonzero \ge \frac{6((D+1)(n'-1)-dn')^2}{D(D+1)(2D+1)}.$$

The proof of the proposition is given in Appendix C.

The proposition implies that for many practical graphs, SimRank similarity function contains more non-zero values than is actually needed for general applications, and more than can be computed using reasonable storage size. In particular, it is typical for scale-free graphs to have their largest strongly connected component contain most of the graph nodes: $n' \sim 0.9n$, and to have the so-called *ultrasmall* diameter [5]: $d \sim \log\log n'$. Consequently, even large scale-free graphs have their diameter less than 5, the typical number of SimRank iterations performed. For $d < 5$, the lower bound established in Proposition 7 becomes:

$$\#\text{nonzero} \ge \frac{(n'-6)^2}{55}, \tag{16}$$

only around 55 times less than having every node similar to every other node. The formula (16) implies that even for a moderate graph with $n' = 3M$ and $d < 5$, SimRank iterative computation produces at least 163 billion non-zero similarity scores after five iterations. Even with 8 bytes per score, computing this iterative similarity function involves processing and storing more than a terabyte of memory.

Since scale-free graphs constitute the underlying representation for many practical corpora including Wikipedia [32] and the Web, we propose the notion of a threshold-sieved similarity function for effectively handling desired similarity scores. The necessary theoretical results are presented to ensure that a threshold-sieved similarity function provides a user-controlled effect over the result similarity scores.

Let us choose some non-negative parameters $\delta_1, \delta_2, \ldots, \delta_K$, where each δ_k is treated as a threshold for iterative similarity scores on the kth iteration. Conceptually, similarity score for a pair of nodes a, b on the kth iteration will be treated as zero if both (1) this value is not greater than the threshold δ_k and (2) similarity between a and b is zero on the previous iteration recursively.

Formally, let us define a threshold-sieved iterative similarity function $R_k^{\delta_k}(*,*)$ over a set of threshold parameters $\{\delta_k\}$ as follows:

$$R_0^{\delta_0}(a,b) = R_0(a,b);$$

$$R_{k+1}^{\delta_{k+1}}(a,a) = R_{k+1}(a,a) = 1; \tag{17}$$

$$R_{k+1}^{\delta_{k+1}}(a,b) = 0, \text{ if } I(a) = \emptyset \text{ or } I(b) = \emptyset; \tag{18}$$

$$R_{k+1}^{\delta_{k+1}}(a,b) = \frac{C}{|I(a)|\,|I(b)|}\sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|} R_k^{\delta_k}(I_i(a), I_j(b)),$$
$$\text{if either (right-hand side} > \delta_{k+1}\text{)}$$
$$\text{or } R_k^{\delta_k}(a,b) \ne 0; \tag{19}$$

$$R_{k+1}^{\delta_{k+1}}(a,b) = 0, \text{ otherwise.} \tag{20}$$

In $R_{k+1}^{\delta_{k+1}}(a,b)$ definitions (18)–(20), a and b are assumed to be different nodes; when a and b are the same node, the definition of $R_{k+1}^{\delta_{k+1}}(a,a)$ is given separately in (17).

For textually distinguishing $R_k(*,*)$ and $R_k^{\delta_k}(*,*)$, we will further refer to $R_k(*,*)$ as *conventional* iterative similarity.

It can easily be proven by mathematical induction that conventional similarity function is an upper bound for a threshold-sieved one, i.e.

$$R_k^{\delta_k}(a,b) \le R_k(a,b), \quad \forall a,b, \quad \forall k. \tag{21}$$

Moreover, the following estimate for threshold-sieved iterative similarity function with respect to conventional iterative similarity function can be established:

Proposition 8 *For every iteration* $k = 0, 1, 2, \ldots$ *and for every two nodes* $a, b \in V$ *the following estimate holds:*

$$R_k(a,b) - R_k^{\delta_k}(a,b) \le \Delta,$$

where

$$\Delta = \sum_{m=1}^{k} C^{k-m}\delta_m. \tag{22}$$

The proof of the proposition is given in Appendix D.

The parameter Δ is intended as a user control over maximum potential difference between threshold-sieved and conventional iterative similarity functions, and thus Δ is generally chosen by a user. Provided that $\delta_1, \delta_2, \ldots, \delta_k$ are selected to fulfill (22), Proposition 8 states that difference between threshold-sieved and conventional similarity scores does not exceed Δ.

Note from the proposition proof that the Eq. (22) gives the worst-case upper bound. In practice, we noted the differences between $R_k(*,*)$ and $R_k^{\delta_k}(*,*)$ for $k = 5$ being smaller than Δ by one order of magnitude.

It should be noted that if Δ is chosen to be zero, then $\delta_1 = \delta_2 = \ldots = \delta_K = 0$ and a threshold-sieved iterative similarity function $R_k^{\delta_k}(*,*)$ becomes a conventional iterative similarity function $R_k(*,*)$. From this perspective, a threshold-sieved similarity can be considered as a generalization for a conventional similarity.

One of the possible ways for choosing threshold parameters $\delta_1, \delta_2, \ldots, \delta_k$ is to specify every iteration make an equal contribution to the Δ value:

$$C^{k-m}\delta_m = \frac{\Delta}{k}, \quad m = \overline{1,k},$$

which gives

$$\delta_m = \frac{\Delta}{kC^{k-m}}, \quad m = \overline{1,k}. \tag{23}$$

Due to general commonalities between a threshold-sieved and a conventional similarity functions, previously stated Propositions 3 and 4 straightforwardly apply to threshold-sieved similarity function. With partial sums introduced in Sect. 4.2, equations (19) and (20) are rewritten correspondingly as:

$$R_{k+1}^{\delta_{k+1}}(a,b) = \frac{C}{|I(a)|\,|I(b)|}\sum_{j=1}^{|I(b)|}\mathrm{Partial}_{I(a)}^{R_k^{\delta_k}}(I_j(b)),$$
$$\text{if either (right-hand side} > \delta_{k+1})$$
$$\text{or } R_k^{\delta_k}(a,b) \neq 0; \tag{24}$$
$$R_{k+1}^{\delta_{k+1}}(a,b) = 0, \text{ otherwise.} \tag{25}$$

In the similar way, equations for cross summation are rewritten as:

$$R_{k+1}^{\delta_{k+1}}(a,b) = \frac{C}{|I(a)|\,|I(b)|}\sum_{j=1}^{|I(b)|}\mathrm{Outer}_{I}^{R_k^{\delta_k}}(a, I_j(b)),$$
$$\text{if either (right-hand side} > \delta_{k+1})$$
$$\text{or } R_k^{\delta_k}(a,b) \neq 0; \tag{26}$$
$$R_{k+1}^{\delta_{k+1}}(a,b) = 0, \text{ otherwise.} \tag{27}$$

The combination of Propositions 1 and 8 provides the upper bound for a maximum potential difference between a threshold-sieved similarity function and the theoretical similarity function:

Proposition 9 *For every pair of nodes (a,b) and for every iteration number $k = 0, 1, 2, \ldots$, the following estimate holds:*

$$s(a,b) - R_k^{\delta_k}(a,b) \leq \epsilon,$$

where

$$\epsilon = C^{k+1} + \Delta, \tag{28}$$

and threshold parameters $\delta_1, \delta_2, \ldots, \delta_k$ are chosen with respect to Δ as specified by (22).

Proof The proposition immediately follows from Propositions 1 and 8. □

In combination with (21) and SimRank Property 2, the estimate stated in Proposition 9 has the following form:

$$0 \leq s(a,b) - R_k^{\delta_k}(a,b) \leq \epsilon.$$

Based on Proposition 9, the analogue of Proposition 2 for ranking accuracy estimate can be straightforwardly stated for a threshold-sieved similarity function, replacing C^{k+1} with ϵ.

A threshold-sieved iterative similarity function $R_k^{\delta_k}(*,*)$ may at first seem as an approximation for a conventional iterative similarity function $R_k(*,*)$; however, Proposition 9 shows that both actually follow the same nature of uniformly converging to the theoretical similarity function $s(*,*)$ for $k \to \infty$, $\Delta \to 0$. When a user wishes to achieve a desired accuracy ϵ, she or he is free to follow one of the two possible options. The user can turn threshold sieving off by specifying $\Delta = 0$, which in accordance with (28) would result in a fewer number of iterations K required for achieving accuracy ϵ:

$$K = \lceil \log_C \epsilon \rceil - 1.$$

Alternatively, the user can perform one more iteration and assign additional accuracy tolerance to Δ:

$$K = \lceil \log_C \epsilon \rceil;$$
$$\Delta = \epsilon - C^{K+1}.$$

As our experimental results presented in Sect. 7 show, the latter option provides a faster computation due to a more freedom for applying the suggested optimization techniques, even though the additional SimRank iteration is performed.

5 Algorithm for SimRank optimized computation

In this section, the optimization techniques presented above are integrated into the general SimRank optimized computation algorithm.

As it was discussed in Subsects. 4.2 and 4.3, partial sums and cross summation can be used as alternative techniques for organizing SimRank optimized iteration. Consequently, two alternative algorithms are presented that implement SimRank iteration with respect to each of the techniques. In different cases, one or the other algorithm is preferable depending on input data involved; the discussion on this subject is given in Sect. 7.

5.1 SimRank iteration using partial sums

Algorithm 1 implements SimRank optimized iteration using essential node pairs, partial sums, and threshold-sieved similarities. The algorithm accepts as input: the graph G, the decay factor C, the iterative similarity function from the previous iteration $R_k^{\delta_k}(*,*)$, and the threshold parameter δ_{k+1}. The algorithm returns as output the iterative similarity function $R_{k+1}^{\delta_{k+1}}(*,*)$ computed as the result of this iteration.

Algorithm 1 SimRank iteration using partial sums

Input: $G(V, E)$, C, $R_k^{\delta_k}(*,*)$, δ_{k+1}
Output: $R_{k+1}^{\delta_{k+1}}(*,*)$
1: **for all** $a \in V$ **do**
2: **if** $O(a) = \emptyset$ and $k \neq K-1$ **then**
3: Continue for next a
4: **end if**
5: Initialize $\text{Partial}_{I(a)}^{R_k}(*)$
6: Calculate $\text{Essential}_{R_k}(a)$
7: **for all** $b \in \{a\} \cup \text{Essential}_{R_k}(a)$ **do**
8: Calculate $R_{k+1}^{\delta_{k+1}}(a,b)$
9: **end for**
10: Free $\text{Essential}_{R_k}(a)$
11: Free $\text{Partial}_{I(a)}^{R_k}(*)$
12: **end for**

As algorithm statements are described in high-level terms, the following list collects references to underlying formulas and theoretical justifications given earlier in the paper and implied in a corresponding algorithm line:

- In lines 5.1–5.1, the conditional expression is justified by Corollary 1.
- In line 5.1, $\text{Essential}_{R_k}(a)$ is calculated by (10), (11).
- In line 5.1, the condition in the header of the for loop is justified by Proposition 3.
- In line 5.1, $R_{k+1}^{\delta_{k+1}}(a,b)$ is calculated by (17), (18), (24) and (25).
- In lines 5.1 and 5.1, the free statement is used to denote that $\text{Essential}_{R_k}(a)$ and $\text{Partial}_{I(a)}^{R_k}(*)$ are not required for further computation and can be dropped.

5.2 SimRank iteration using cross summation

Cross summation involves outer summation followed by inner summation; Algorithm 2 gives the detailed description of the outer one:

- In line 5.2 of the algorithm, the value of the outer sums function is calculated in accordance with Eq. (15).
- In line 5.2, the free statement is used to denote that the values of the function $\text{Combination}_{R_k(*,b)}^{j}(*)$ are not required for subsequent computation and can be dropped.

Algorithm 2 Outer summation

Input: $G(V, E)$, $\{V_i\}$, $\text{Bucket}_I^{\{V_i\}}(*,*)$, $R_k^{\delta_k}(*,*)$
Output: $\text{Outer}_I^{R_k}(*,*)$
1: **for all** $b \in V$ **do**
2: Initialize $\text{Outer}_I^{R_k}(*,b) = 0$
3: **for all** V_j, $j = \overline{1,q}$ **do**
4: Calculate $\text{Combination}_{R_k(*,b)}^{j}(*)$
5: **for all** $a \in V$ **do**
6: Increment: $\text{Outer}_I^{R_k}(a,b)$ += += $\text{Combination}_{R_k(*,b)}^{j}(\text{Bucket}_I^{\{V_i\}}(a,j))$
7: **end for**
8: Free $\text{Combination}_{R_k(*,b)}^{j}(*)$
9: **end for**
10: Write $\text{Outer}_I^{R_k}(*,b)$
11: **end for**

The algorithm for inner summation is very similar to the one for outer summation and is thus omitted. Technically, inner summation has the following changes with respect to outer summation:

1. Inner summation accepts $\text{Outer}_I^{R_k}(*,*)$ instead of $R_k^{\delta_k}(*,*)$ as input and returns $R_{k+1}^{\delta_{k+1}}(*,*)$ as the result;
2. The relative nesting of the for-loops over nodes a and b is swapped;
3. In the end of inner summation, each computed value is normalized by $\frac{C}{|I(a)|\cdot|I(b)|}$ and is compared against the threshold parameter δ_{k+1}.

Combining outer and inner summations, Algorithm 3 implements SimRank iteration using cross summation. The algorithm accepts and returns the same parameters as Algorithm 1 discussed above.

Algorithm 3 SimRank iteration using cross summation

Input: $G(V, E)$, C, $R_k^{\delta_k}(*,*)$, δ_{k+1}
Output: $R_{k+1}^{\delta_{k+1}}(*,*)$
1: Choose $\{V_i\}$, $i = \overline{1,q}$
2: Calculate $\text{Bucket}_I^{\{V_i\}}(*,*)$
3: $\text{Outer}_I^{R_k}(*,*)$ = outer_summation($G(V, E)$, $\{V_i\}$, $\text{Bucket}_I^{\{V_i\}}(*,*)$, $R_k^{\delta_k}(*,*)$)
4: $R_{k+1}^{\delta_{k+1}}(*,*)$ = inner_summation($G(V, E)$, $\{V_i\}$, $\text{Bucket}_I^{\{V_i\}}(*,*)$, $\text{Outer}_I^{R_k}(*,*)$, C, δ_{k+1})

5.3 The complete iterative computation

Having introduced the above lower-level algorithms that implement a single SimRank iteration, the complete iterative computation is organized as a compact Algorithm 4. As different strategies for choosing K and Δ for achieving the desired accuracy can be applied, the algorithm accepts both parameters as input.

Algorithm 4 SimRank optimized computation

Input: $G(V, E)$, C, K, Δ
Output: $R_K^{\delta_K}(*,*)$
1: Calculate $\delta_1, \delta_2, \ldots, \delta_K$
2: Initialize $R_0(*,*)$
3: **for** $k = 0$ to $K - 1$ **do**
4: $\quad R_{k+1}^{\delta_{k+1}}(*,*) = \text{iteration}(G(V, E), C, R_k^{\delta_k}(*,*), \delta_{k+1})$
5: **end for**

The statements involved in Algorithm 4 are elaborated as follows:

- In line 5.3, each δ_m can be calculated by equation (23). A different strategy for choosing $\delta_1, \delta_2, \ldots, \delta_K$ can be used; the only requirement is that (22) be fulfilled.
- In line 5.3, $R_0(*,*)$ is defined by (2).
- In line 5.3, $R_{k+1}^{\delta_{k+1}}(*,*)$ is calculated as the result of performing a single iteration. For an iteration, either partial sums described in Algorithm 1 or cross summation described in Algorithm 3 can be used.

Note that within the same iterative computation, different iterations can rely on different algorithms—either on Algorithm 1 or on Algorithm 3. Recommendations on choosing a particular optimization algorithm for a given iteration are provided in Sect. 7.

Collecting complexity analysis for all suggested optimization techniques, it follows that a single SimRank iteration requires $O(nl)$ operations in the worst case for Algorithm 1 and $O(\frac{n^3}{\log_2 n})$ operations for Algorithm 3.

The complete iterative process consequently has the computational complexity of $K \cdot \min(O(nl), O(\frac{n^3}{\log_2 n}))$. As it was shown in Propositions 1 and 9 that the number of iterations K required for achieving the desired accuracy does not depend on the number of nodes n in the graph, we finally obtain that SimRank computation in the presence of the suggested optimization techniques is $\min(O(nl), O(\frac{n^3}{\log_2 n}))$.

6 Related work

Due to practical importance of measuring object-to-object similarity, different approaches to defining similarity measures were suggested in scientific literature, e.g. the ones based on domain hierarchies [9], information theory [18], network flow computation [22]. With respect to the focus of this paper, a detailed discussion is given to related work that either correlate with SimRank or present similarity measures with complexity analysis claimed applicable for large data corpora.

Xi et al. suggested a similarity-calculating algorithm called *SimFusion* that aims at "combining relationships from multiple heterogeneous data sources" [31]. The basic intuition behind SimFusion approach somewhat resembles the one for SimRank: "the similarity between two data objects can be reinforced by the similarity of related data objects from the same and different spaces" [31]. SimFusion provides the following extensions with respect to SimRank: (1) support for different kinds of intra-nodes relations, e.g. outgoing links, content commonality; (2) support for different weights associated with different kinds of relations; (3) support for several information spaces.

Iterative similarity computation formula for SimFusion has much in common with the one for SimRank. Indeed, with L_{urm} being a row-stochastic matrix that combines all the relationships between nodes, SimFusion reinforcement assumption is reified as follows:

$$S_{\text{usm}}^k = L_{\text{urm}} S_{\text{usm}}^{k-1} L_{\text{urm}}^T, \tag{29}$$

where S_{usm} is a "unified similarity matrix" that represents similarity values between node pairs [31]. Let us denote a row in L_{urm} that corresponds to node a as $L_{\text{urm}}(a)$, and a matrix element in S_{urm}^k that corresponds to a pair of nodes (a, b) as $S_{\text{usm}}^k(a, b)$. If we then consider node relations of the kind "has an incoming link" and treat all incoming links as of an equal priority, then $L_{\text{urm}}(a)$ contains $\frac{1}{|I(a)|}$ in column $I_i(a)$, $i = \overline{1,n}$ and zeroes in all remaining columns. In accordance with (29), SimFusion iterative similarity value between nodes a and b thus takes the form:

$$S^k_{\text{usm}}(a,b) = L_{\text{urm}}(a) S^{k-1}_{\text{usm}} (L_{\text{urm}}(b))^T$$
$$= \frac{1}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} S^{k-1}_{\text{usm}}(I_i(a), I_j(b)), \quad (30)$$

which is the same as for SimRank iterative formula minus the decay factor C. Unlike SimFusion, SimRank similarity score for any node with itself always equals to 1, and these initial similarity scores are iteratively propagated to the other node pairs. In SimFusion, initial similarity scores are redistributed in a flow fashion through node relations, and thus node similarity with itself may not be equal to 1.

If a node has no incoming links, it has zero SimRank similarity score with every other node except for itself. In SimFusion, a node with no relationship to the other nodes in data space has each element in the corresponding row of relationship matrix set to $1/n$ for preventing similarity sinks. This treatment has the same effect as introducing a source of rank in PageRank [26].

In spite of the noted differences between SimRank and SimFusion, Eq. (30) shows that the overall iterative formulas for SimRank and SimFusion have much resemblance to each other. Consequently, some of the optimization techniques presented in this paper should apply to SimFusion computation as well, e.g. essential node pairs selection in terms of sparse matrices. We also believe that there should exist an analogous accuracy estimate for SimFusion as the one revealed for SimRank.

The computational complexity for SimFusion is $O(Knl)$ that directly follows from the matrix representation (29). For comparison, cross summation we proposed in this paper provides computational complexity $O(\frac{n^3}{\log_2 n})$, being more computationally efficient than $O(nl)$ for an input graph with a large proportion of links. SimRank similarity scores can thus be computed more efficiently than SimFusion ones, since cross summation is not directly applicable to a row-stochastic matrix.

Fogaras and Rácz [7] suggested a scalable framework for SimRank computation based on Monte Carlo method. The main idea of their approach is to generate reversed random walks for each node in a graph, calculate the first meeting time $\tau_{a,b}$ for a pair of random walks started in nodes a and b, and estimate $s(a,b)$ by $C^{\tau_{a,b}}$. In their work, Fogaras and Rácz suggest several excellent ideas, in particular, fingerprint trees, random permutations on graph nodes for effectively generating coupled random walks, parallelization possibilities for SimRank computation under their framework. The probabilistic approach they took allowed them to significantly reduce the computational complexity and to create a framework for similarity computation scalable enough for performing SimRank precomputation phase for a graph with 79M nodes. Fogaras and Rácz provide a solid theoretical basis for approximations they make and a (probabilistic) error estimate for their Monte Carlo similarity function. Valuable experimental results are obtained for path length and the decay factor value; surprisingly however, Monte Carlo similarity was not compared with iterative SimRank similarity from the perspective of scores quality. The differences in our approaches are that Fogaras and Rácz initially base their framework on Monte Carlo method, and thus their computation is inherently probabilistic, whereas our work is focused on iteratively computing the exact similarity scores.

For improving time and space requirements for SimRank computation, Jeh and Widom [12] suggested pruning the logical graph G^2. Their proposal is to "set the similarity between two nodes far apart to be 0, and consider node-pairs only for nodes which are near each other" [12]. This technique is based on the assumption that "it is very likely that the neighborhood (say, nodes within a radius of 2 or 3) of a typical node will be a very small percentage ($< 1\%$) of the entire domain" [12]. First, this assumption does not hold for scale-free graphs, as these have a very small average distance (or diameter) between nodes [5]. As our early experimental studies over practical scale-free graphs showed, even the neighbourhood within the radius of 2 contains a considerable proportion of graph nodes.

Second, pruning graph G^2 in the suggested way is an approximation; Jeh and Widom provide no theoretical argument about the error of approximating [7], but admit that "the quality of the approximation needs to be verified" [12]. Finally, removing node pairs not near each other from consideration undermines the basic SimRank design principle of being a generalization to a conventional immediate in-neighbours analysis [28]. As verified by Fogaras and Rácz in their experiments, "the multi-step neighborhoods of pages[2] contain valuable similarity information" [7].

For comparison, optimization techniques suggested in this paper are free from the above mentioned drawbacks. Selecting essential node pairs does not affect iterative similarity scores. Threshold sieving has a controllable effect over iterative scores precision that can be restored by performing an additional iteration. The techniques do not decrease the radius of a node neighbourhood considered for similarity scores computation.

For making SimRank similarity scores more intuitive for the area of sponsored search, Antonellis et al. [3] suggested the similarity measure "Simrank++" by extending SimRank with two enhanced versions. The enhanced versions make similarity scores include evidence factor of incident nodes and link weights information. Although some of the optimization techniques suggested here are applicable to Simrank++ as well, link weights added by Antonellis et al. make

[2] The term "page" there is used in the same semantics as the term "node" throughout this paper.

computational complexity of Simrank++ be no better than $O(nl)$ in the general case.

In order to make Simrank++ practically computable for a bipartite graph with totally 3.2M nodes and 4M links, Antonellis et al. suggested splitting the graph into several smaller subgraphs by finding cuts with small conductance [2]. While splitting a graph into multiple subgraphs can be a promising approach for speeding up SimRank computation, the effect of considering subgraphs separately on the accuracy of computed SimRank scores remained outside the scope of consideration by Antonellis et al.

Maguitman et al. [23] introduce an information-theoretic measure of *semantic similarity* that exploits human-generated topical directories metadata and relies on "both hierarchical and non-hierarchical structure of an ontology" [23]. Maguitman et al. suggest a flexible mechanism for extending the previously existing tree-based semantic similarity measures to a graph-based semantic similarity; however, their results are based on the assumption that a graph necessarily has a "hierarchical (tree) component" T.

Maguitman et al. claim computational complexity $O(n^3)$ for their semantic similarity measure for n topics [23]; however, that is the computational complexity for just a single matrices product $A \odot B$ used in their reasoning. Precisely, semantic similarity measure requires the closure matrix T^+ computed; the latter is defined as $T^+ = \lim_{r\to\infty} T^{(r)}$, with $T^{(r+1)} = T \odot T^{(r)}$, $T^{(0)} = T$. Consequently, obtaining T^+ alone implies computational complexity $O(n^3 h)$, where h is "the maximum depth of the tree T" [23]. Generally, h depends on n, and the worst case the computational complexity for T^+ is actually $O(n^4)$, not $O(n^3)$.

Maguitman et al. required significant computational and storage resources for computing similarity scores for their semantic similarity measure for a data corpus consisting of 0.5M topics containing totally 1.23M pages [23]. For comparison, our claim is that SimRank optimization techniques suggested in this paper allow computing SimRank similarity scores on a commodity desktop machine in reasonable time even for a larger data corpora, as our experimental results illustrate.

Using once computed semantic similarity scores as a baseline, Maguitman et al. introduce and compare several approximation measures. Notably, link-based similarity measures systematically produced better correlation with semantic similarity measure and correspondingly with human judgments compared to text-based measures [23]. This important result can in particular serve as an approval for SimRank being a purely link-based similarity measure.

Lin et al. [19] suggest a similarity measure based on PageRank scores propagation through link paths. With r standing for the propagation radius and d for an average node degree, finding similar nodes to a given node with respect to that measure has computational complexity of $O(d^{2r})$, which we believe being too ineffective for on-line computation assumed in [19].

Geerts et al. [10] introduce the concept of a database graph for expressing relationships between partial tuples in a relational database and explore methods for ranking partial tuples. Defining similarity measures in a database graph is pointed out as an interesting question for future work. We believe that SimRank can be a candidate measure for this domain, since a database graph contains enough information for computing SimRank similarity scores for partial database tuples, and the suggested optimization techniques provide viable computational complexity.

7 Experiments

In this section, experimental results are presented for illustrating the practical quantitative effect of applying the optimization techniques presented in this paper.

We implemented a prototype that provides SimRank similarity scores computation and incorporates the suggested optimization techniques in accordance with Algorithm 4 given in Sect. 4. From the implementation perspective, each graph node is identified by a distinct non-negative integer. Iterative SimRank similarity function for each iteration is represented by a square matrix, with a matrix element standing for the similarity score between nodes identified by the corresponding row and column numbers. From the storage perspective, a matrix is sparse, in that constant 1s across the main diagonal and zero similarity scores are not stored. Matrix storage in external memory is implemented on top of Oracle Berkeley DB.[3] As noted in Subsect. 4.2, the suggested optimization techniques facilitate clustering similarity function values by the first argument, corresponding to clustering a matrix by rows in matrix terms. For each row number, the associated (column_number, similarity_score) pairs are stored adjacently. Each of the mappings $I(v)$ and $O(v)$ is implemented as an association between a node identifier for v and a list of node identifiers for the corresponding in-neighbours and out-neighbours, respectively.

Four kinds of experiments are reported here, each in its own subsection. The first experiment investigates SimRank computation time for different optimization techniques employed. The second experiment compares partial sums and cross summation and provides the recommendation for a preferable technique in each case. The third experiment investigates the number of non-zero similarity scores on each SimRank iteration with respect to graph diameter and threshold sieving. Finally, the last subsection reports our experience

[3] http://www.oracle.com/technology/products/berkeley-db/index.html.

Table 1 SimRank computation time w.r.t. the number of nodes in a graph for different optimization techniques configurations

For each subsequent column, another optimization technique is turned on. For the last column, $\Delta < C^6 - C^7$

Number of nodes in a graph	Computation time, seconds			
	No optimization	Selecting essential node pairs	Turn on Partial sums	Turn on Threshold sieving, $K = 6$
1000	42	11	2	2
2000	348	157	25	13
5000	8061	5181	588	309
10000	165902	131675	8145	2799

in computing SimRank scores over the English Wikipedia corpus.

7.1 SimRank computation time

The purpose of this experiment is to investigate the dynamics in SimRank computation time with respect to the number of nodes and links in a graph and particular optimization techniques employed. We have chosen scale-free graphs for the experiment, because a node in a scale-free graph generally has non-zero similarity scores with a significant proportion of other nodes after several SimRank iterations (recall Proposition 7), rather than with a fixed number of nodes. Additionally, the number of incident links for each node in a graph was made proportional to the total number of nodes, thus making $l \sim O(n^2)$. These settings of the experiment were chosen in order to investigate a pessimistic case in SimRank computation, rather than an optimistic case.

The set of generated graphs was produced by the scale-free graph generator.[4] For the purposes of the experiment, the implementation was made configurable, with the ability of turning each optimization technique on and off at compile-time. For properly taking access operations time into account, unbiased by cache speed, the cache size in Oracle Berkeley DB was chosen proportionally to the number of nodes in a graph and sufficient for keeping just several matrix rows. The following machine configuration was used: 2.1 GHz Intel Pentium processor, 1 Gb RAM and Linux OS.

The averaged computation time with respect to the number of nodes in a graph and the particular optimization techniques used is shown in Table 1. For a correspondence with experiment conditions performed by Jeh and Widom [12], the decay factor C was chosen as 0.8; the number of iterations K was set to 5 for all table columns except for the last one. First, computing SimRank with no optimization, optimization techniques are consequently turned on one by one for each subsequent column in Table 1, thus illustrating the speedup achieved by each individual optimization technique. For the last column, six iterations were performed instead of five, with threshold sieving turned on for Δ chosen as $\Delta = 0.05 < C^6 - C^7$, which in accordance with Proposition 9 gives the same total accuracy of computed similarity scores as for all the remaining columns.

Two important conclusions can be drawn from the experimental results presented in Table 1. First, even for a relatively small graph sizes considered, using partial sums exposes radical speedup that fully corresponds with the theoretical expectations presented in Sect. 4. (Actually, proceeding the comparison for larger input graphs was impractical, since SimRank computation for a graph with 10K nodes took 36+ h in the absence of partial sums). Second, SimRank computation benefits from using threshold-sieved similarity functions, even with the additional iteration performed for recovering the total accuracy of similarity scores. The latter result justifies the recommendation made in the end of Subsect. 4.4.

For graphically illustrating the effect of introducing partial sums for SimRank computation, Fig. 2 shows computation time with respect to the number of nodes in a graph in a more detail in the form of bar charts, for SimRank computed (a) without and (b) with partial sums. For illustrating the correlation between the computation time and the number of nodes in a graph, each bar chart is approximated by a polynomial curve. Note that different scale is chosen across the vertical axis in Fig. 2 (a) and (b) for providing a more illustrative look for each curve shape.

Constants P and Q in Fig. 2 were calculated in accordance with minimum mean square error estimator; quartic and cubic functions, respectively, showed the best approximation for SimRank computation time without and with partial sums, compared to the other polynoms in the number of nodes in a graph. This experimental result fully agrees with the theoretical considerations made in Sect. 4.

[4] Dreier, D. Manual of Operation: Barabasi Graph Generator v1.0. University of California Riverside, Department of Computer Science. (2002).

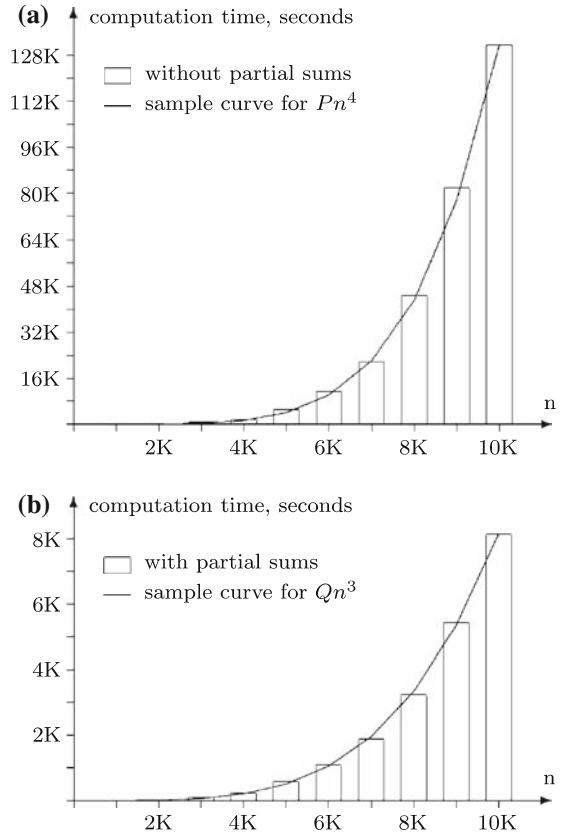

Fig. 2 SimRank computation time with (**b**) and without (**a**) partial sums usage w.r.t. the number of nodes n in a graph

The experiment did not involve cross summation, because it exhibited computation time similar to partial sums in these settings. Cross summation and partial sums are specially compared in the next subsection.

7.2 Partial sums versus cross summation

As we noted in Sect. 4, partial sums together with essential node pairs allow saving much computational effort in case of iterative similarity function containing a relatively small fraction of non-zero values. On the other hand, cross summation is based on eager evaluation and thus the running time of the technique should be quite stable to the number of non-zero values. One can thus expect that partial sums would be computationally preferable than cross summation for an iterative similarity function with fewer non-zero values, and the situation would be the opposite if the function has many non-zero values.

To justify these observations experimentally, we used graphs of different size and connectivity in order to obtain similarity functions with different proportions of non-zero values. In addition, since the number of non-zero values of an iterative similarity function increases in the number of iterations, we measured computation time for each iteration individually.

Table 2 shows SimRank computation time for each iteration for different input graphs, and the corresponding number of non-zero values of the iterative similarity function. It can be observed from the table that the theoretical considerations stated above fully apply in practice: iteration based on partial sums computes faster for an iterative similarity function with small proportion of non-zero values, but computation time increases considerably as the proportion of non-zero values increases. On the other hand, iterative computation based on cross summation exhibits its running time increasing only insignificantly.

For each input graph in Table 2, bold font is used to emphasize a first iteration that completes faster if computed using cross summation than if computed using partial sums. It can be observed from the last column in the table that cross summation is computationally preferable if the proportion of non-zero values of the iterative similarity function roughly exceeds 0.003. It can thus be recommended to begin the iterative computation using partial sums (Algorithm 1) and switch to cross summation (Algorithm 3) starting from an iteration that is expected to have the proportion of non-zero similarity scores exceeding 0.003.

7.3 Counting non-zero similarity scores

This subsection experimentally verifies the lower bound for the number of non-zero SimRank similarity scores established in Proposition 7. For the experiment, we generated a graph with diameter $d = 4.7$ satisfying the pre-conditions of the proposition. Table 3, left side, displays the number of non-zero similarity scores for the graph after each SimRank iteration. It can be seen from the table that the proportion of non-zero scores after five iterations (emphasized in bold) indeed satisfies the inequality (16) stated by Proposition 7.

In fact, the database file in the experiment took up 58+ Gb memory for storing the iterative similarity function after the fifth iteration, so the sample graph was essentially the largest one that can viably be used in practice if all non-zero similarity scores are kept.

The right side of Table 3 shows the number of non-zero similarity scores for the same graph in case of threshold sieving turned on. To preserve the same accuracy of computed scores, we took $\Delta = C^6 - C^7$ and made the additional sixth iteration. It can be seen from the table that threshold-sieved similarity requires storing considerably fewer non-zero scores while achieving the same accuracy. Moreover, even with the lowest similarity scores filtered out, the threshold-sieved similarity function leaves 3,000+ top-similar nodes for each node in the graph on average, which we expect to be sufficient for the needs of most applications.

Table 2 Computation time for each SimRank iteration using partial sums or cross summation for different input graphs

n	l	k	Computation time, seconds		Number of nonzero values	
			Partial sums	Cross summation	Absolute	Relative
50,000	100,000	1	24	879	99,868	$0.00004 \cdot n^2$
		2	26	919	494,351	$0.00020 \cdot n^2$
		3	48	921	1,859,857	$0.00074 \cdot n^2$
		4	262	941	4,807,831	$0.00192 \cdot n^2$
		5	**994**	**958**	8,218,846	$\mathbf{0.00329} \cdot n^2$
30,000	300,000	1	87	324	1,456,250	$0.00162 \cdot n^2$
		2	**2,132**	**597**	13,640,653	$\mathbf{0.01516} \cdot n^2$
		3	37,182	912	32,545,578	$0.03616 \cdot n^2$
50,000	500,000	1	38	1,440	4,247,429	$0.00170 \cdot n^2$
		2	**5,781**	**1,730**	101,211,393	$\mathbf{0.04048} \cdot n^2$
		3	122,617	2,981	315,935,956	$0.12637 \cdot n^2$

Table 3 Number of non-zero similarity scores after each iteration for a graph with $n = 150,000$ nodes, diameter $d = 4.7$

k	Number of non-zero similarity scores			
	$\Delta = 0$		$\Delta = C^6 - C^7$	
	Absolute	Relative	Absolute	Relative
1	12,760,971	$0.0005 \cdot n^2$	9,246,356	$0.0004 \cdot n^2$
2	404,909,492	$0.0180 \cdot n^2$	96,245,954	$0.0043 \cdot n^2$
3	1,579,789,355	$0.0702 \cdot n^2$	192,236,054	$0.0085 \cdot n^2$
4	2,737,206,126	$0.1216 \cdot n^2$	287,262,953	$0.0128 \cdot n^2$
5	3,602,356,569	$\mathbf{0.1601} \cdot n^2$	378,246,595	$0.0168 \cdot n^2$
6			467,356,216	$0.0208 \cdot n^2$

7.4 Experiment over Wikipedia corpus

Our practical interest for implementing the suggested optimization techniques was to compute SimRank over the complete set of articles from English Wikipedia corpus.

Wikipedia is an open content online encyclopedia project that is "created in a collaborative effort of voluntary contributors".[5] Wikipedia is available in many languages, with the English version being the largest one, containing 2.2M articles. In addition to being a popular online encyclopedia, Wikipedia has recently obtained a big academic interest as an information corpus by itself,[6] e.g. [8,30]. However, to the best of our knowledge, nobody has yet reported the experience in computing SimRank scores for English Wikipedia corpus.

Since each Wikipedia article is generally dedicated to describe a single encyclopedic concept, we have naturally chosen an individual article to be a node in the SimRank model. As Wikipedia articles are organized into categories being articles themselves, we chose the relationship "a category contains an article" to be a link from the category to the article. We will further refer to a thus built graph as the Wikipedia graph. Computing SimRank scores over the Wikipedia graph has the semantics of obtaining similarity scores for encyclopedic concept pairs. Note that the Wikipedia graph covers only a subset of Wikipedia corpus, since category links constitute a subset of links available in Wikipedia.

As we had a practical interest in computing SimRank over the Wikipedia graph from the beginning of our research, our implementation evolved throughout the optimization techniques development. Before any optimization techniques were introduced, preliminary experiments showed that SimRank computation over the Wikipedia graph would have taken months at least to complete, which was unacceptable.

After the set of optimization techniques presented in this paper has been worked out, SimRank computation over the Wikipedia graph takes approx. 17 h to complete on a single machine, making it possible to perform the computation on a nightly basis. The following SimRank computation

[5] http://wikipedia.org/.

[6] Wikipedia in academic studies. http://en.wikipedia.org/wiki/Wikipedia:Wikipedia_in_academic_studies.

parameters are used: $C = 0.6$, $K = 5$, $\Delta = 0.05$. The chosen parameters values provide reasonable accuracy for the computed SimRank scores: $\epsilon = 0.6^{5+1} + 0.05 < 0.1$. Note that due to the polynomial dependence between the decay factor C and accuracy ϵ, even a relatively small decrease in the decay factor results in a considerable improvement in accuracy. Partial sums are used for the experiment for all iterations, because threshold sieving achieves the proportion of non-zero similarity scores small enough to make partial sums computationally preferable over cross summation (recall the previous subsection). We use a machine with 3 GHz Intel Pentium 4 processor, 4 Gb RAM and 32-bit Linux OS; the cache size of 256 Mb is specified for Oracle Berkeley DB.

Note that the Wikipedia graph provides a practical illustration for Corollary 1 presented in Sect. 4.1: Wikipedia graph nodes not being categories have no outgoing links, and thus SimRank scores for them are computed on the last iteration only, while correctly preserving the semantics of the SimRank iterative model.

SimRank similarity scores for Wikipedia concepts pairs provide a valuable practical source of information. We are using the computed scores for extending search engines functionality and for word sense disambiguation. Our further plans include using the computed scores for automatic news feeds classification.

8 Future work

Although optimization techniques presented in this paper provide a considerable improvement to SimRank computational complexity, our profiling experiments revealed that a significant proportion of computation time is occupied by graph access operations $I(v)$ and $O(v)$. Although SimRank computational complexity is guaranteed to remain proportional to $n \cdot l$, graph access operations may become a performance bottleneck in the growing size of input graphs. Our future work thus involves developing further optimization techniques for speeding up graph access operations. Our current vision to achieving scalability in the growing size of input graph is splitting the graph into several (generally, intersecting) subgraphs in such a way that

1. each subgraph could be passed to its own parallel computation instance as if the complete graph without changing the result of similarity scores computation for node pairs processed by that instance; and
2. each subgraph is small enough to fit into main memory for maximizing the speed of access operations.

Our future plans include developing a systematic procedure for splitting a general graph into subgraphs that satisfy the above listed requirements. In particular, candidate algorithms for locating common borders of the subgraphs can probably be minimum-cut/maximum-flow partition algorithm [6] or maximized-modularity algorithm [25]. We also believe that scale-free graphs theory [16] can be exploited for developing the graph split procedure for scale-free graphs, which constitute an underlying model for many existing practical domains.

Some of the algorithms for parallel computation of PageRank could probably be applicable to SimRank computation as well. For PageRank, two different approaches to parallel computation are presented in literature. The first approach is to divide an input graph into blocks [11,24,27] and then to apply conventional methods such as the Jacobi one to these blocks in parallel. Another approach is to approximate PageRank scores by a close but not the exact rank vector [13]. In this case, PageRank is computed for higher-level formations such as web host or web domains as indivisible items. For the internal link structure of these formations, PageRank is computed independently in parallel. The most computationally efficient method presented by Kohlschütter et al. [14] uses a combination of both approaches and relies on host-based link locality which means that web sites contain more internal links than external ones. Based on self-similarity property of complex networks [29], we can suppose that the property of host-based link locality holds for subgraphs of the Web as well. In particular, for Wikipedia this supposition is confirmed by the Wikipedia cluster structure [20].

9 Conclusion

The paper addresses the issues missing for similarity measure SimRank, namely, accuracy estimate and optimization techniques, for facilitating SimRank wider application.

A precise accuracy estimate for SimRank iterative computation is established. The estimate reveals that SimRank computation parameters suggested in the original SimRank proposal implied a relatively low accuracy, and the choice for different parameter values is suggested. The accuracy estimate allows a priori finding out the correct number of iterations required for achieving a desired accuracy. The number of iterations turns out to be independent of input graph characteristics, the fact to benefit scalability.

Optimization techniques are suggested and integrated into the general algorithm to provide a systematic improvement for SimRank computational complexity.

Experimental results show a 50 times speedup achieved by the optimization techniques for a graph with 10K nodes, and relative improvement in computation time further increases for larger graphs. The experience in computing SimRank scores over the English Wikipedia corpus exhibits practical viability of the approach for relatively large data corpora.

We believe that the results presented in the paper would facilitate a wider application of SimRank to computer science techniques, as this similarity measure definitely deserves.

Appendix A: Proof of Proposition 1

Proof (of Proposition 1) If $a = b$ then $s(a, a) = R_k(a, a) = 1$ by definition for every $k = 0, 1, 2, \ldots$, the left-hand side of (4) is zero, and thus (4) obviously holds.

Similarly, if $I(a) = \emptyset$ or $I(b) = \emptyset$ then by definition $s(a, b) = R_k(a, b) = 0$, and the left-hand side of (4) is zero as well.

For the general case of $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$, the proof is organized by mathematical induction.

Induction Basis Let us prove that (4) holds for $k = 0$, i.e. that for every two nodes a, b:

$$s(a, b) - R_0(a, b) \leq C. \tag{31}$$

Since $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$, then $R_0(a, b) = 0$ by definition, $s(a, b)$ is defined by the general recursive equation (1), and consequently

$$\begin{aligned} &s(a, b) - R_0(a, b) = s(a, b) \\ &= \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} \underbrace{s(I_i(a), I_j(b))}_{\leq 1} \\ &\leq \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} 1 = C, \end{aligned}$$

which proves (31).

Inductive step Provided that (4) holds for a given k for all node pairs, let us prove that (4) holds for $(k + 1)$ as well:

$$\begin{aligned} &s(a, b) - R_{k+1}(a, b) \\ &= \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b)) \\ &\quad - \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} R_k(I_i(a), I_j(b)) \\ &= \frac{C}{|I(a)|\,|I(b)|} \\ &\quad \times \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} \underbrace{\{s(I_i(a), I_j(b)) - R_k(I_i(a), I_j(b))\}}_{\leq C^{k+1} \text{ by inductive hypothesis}} \\ &\leq \frac{C \cdot |I(a)|\,|I(b)| \cdot C^{k+1}}{|I(a)|\,|I(b)|} = C \cdot C^{k+1} = C^{(k+1)+1}. \end{aligned}$$

The latter finally proves (4). □

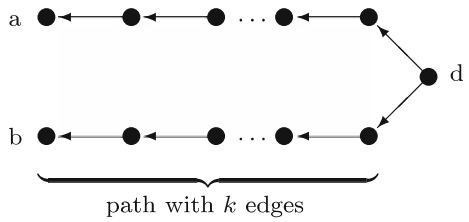

Fig. 3 Illustration of the upper bound stated in Proposition 1 reached: $R_k(a, b) = 0$, $R_{k+1}(a, b) = C^{k+1}$

Appendix B: Example that verifies Note 1

Let us consider an arbitrary $k = 0, 1, 2, \ldots$ and a pair of nodes a, b in a graph presented in Fig. 3 for the chosen k. In this figure, each of the nodes a and b has an incoming directed path of length $(k + 1)$ that starts from some common node d. In such a graph configuration, it can easily be seen that $R_k(a, b) = 0$, whereas $R_{k+m}(a, b) = C^{k+1}$, $m = 1, 2, 3, \ldots$, which gives $s(a, b) = C^{k+1}$. Subtracting $R_k(a, b)$ from $s(a, b)$, we obtain

$$s(a, b) - R_k(a, b) = C^{k+1},$$

which gives the precise upper bound stated in Proposition 1.

Appendix C: Proof of Proposition 7

Proof (of Proposition 7) Since d is the diameter of the strongly connected component, there exists a node x such that the average distance from x to any node in the component does not exceed d. Indeed, if for every node x the average distance from x to all nodes in the strongly connected component were exceeding d, then the average distance between a pair of nodes in the component would have been more than d as well, contradicting the definition of the diameter.

Let us denote the set of all nodes from the strongly connected component as V'. It follows from $d \leq K$ that $d < \infty$, and thus there exists a finite maximum distance p from the node x to any other node in V':

$$p = \max_{u \in V'} \text{distance}(x, u) < \infty.$$

Let N_i denote the set of all nodes from V' that have their distance from x equal to i:

$$N_i = \{u \in V' \mid \text{distance}(x, u) = i\}.$$

For any $i \neq j$, the sets N_i and N_j have an empty intersection; and altogether the sets $N_0, N_1, \ldots, N_p$ cover all nodes of the strongly connected component. Thus, $\sum_{i=0}^{p} |N_i| = n'$. Moreover, as the distance from x to any other node is greater than zero, it follows that $N_0 = \{x\}$, and consequently $|N_0| = 1$ and:

$$\sum_{i=1}^{p} |N_i| = n' - 1. \tag{32}$$

By definition of N_1, all nodes from N_1 have the node x for its common in-neighbour. Consequently, by definition of SimRank iterative computation, all nodes from N_1 have non-zero pair-wise iterative similarity score after the first SimRank iteration. In the same manner, all nodes from N_2 have non-zero iterative similarity scores after the second SimRank iteration; all nodes from N_k have non-zero iterative similarity after kth iteration, for $k = \overline{1, p}$. Consequently,

$$\#\text{nonzero} \geq \sum_{i=1}^{K} |N_i|^2 \geq \{\text{since } K \geq D\} \geq \sum_{i=1}^{D} |N_i|^2. \quad (33)$$

The fact that the average distance from the node x to any other node does not exceed d, can be rewritten using $|N_1|$, $|N_2|$, ..., $|N_p|$ as:

$$\sum_{i=1}^{p} i\,|N_i| \leq \mathrm{d}n'.$$

Let us split the latter summation into three parts:

$$\sum_{i=1}^{D} i\,|N_i| + (D+1)\,|N_{D+1}| + \sum_{i=D+2}^{p} i\,|N_i| \leq \mathrm{d}n' \quad (34)$$

Using the Eq. (32), the value $|N_{D+1}|$ is equal to

$$|N_{D+1}| = n' - 1 - \sum_{i=1}^{D} |N_i| - \sum_{i=D+2}^{p} |N_i|\,.$$

Placing the latter into (34), we obtain:

$$\sum_{i=1}^{D} \overbrace{(i-(D+1))}^{<0} |N_i| + (D+1)(n'-1) + \sum_{i=D+2}^{p} \underbrace{(i-(D+1))}_{>0} |N_i| \leq \mathrm{d}n'.$$

Thus,

$$\sum_{i=1}^{D} \underbrace{(D+1-i)}_{>0} |N_i| \geq \sum_{i=D+2}^{p} (i-D-1)\,|N_i| + (D+1)(n'-1) - \mathrm{d}n'.$$

Since our goal is estimating the lower bound for $\sum_{i=1}^{D} |N_i|^2$, it follows from the latter inequality that the lowest value is achieved for $|N_i| = 0$, $i = \overline{D+2, p}$, and:

$$\sum_{i=1}^{D} (D+1-i)\,|N_i| \geq (D+1)(n'-1) - \mathrm{d}n'. \quad (35)$$

Note that since $d > 2$, the right-hand side of the inequality (35) is always positive. Considering $(|N_1|, |N_2|, \ldots, |N_D|)$ as a D-dimensional vector, the inequality (35) specifies a D-dimensional half-space. Recalling the formula (33), our goal in proving the proposition is to calculate the square distance between the half-space and the point $(0, 0, \ldots, 0)$.

The surface vector for the half-space is: $(D, D-1, D-2, \ldots, 1)$. Thus, the point in the half-space closest to $(0, 0, \ldots, 0)$ is defined by the set of equations:

$$\begin{cases} (|N_1|, |N_2|, \ldots, |N_D|) = (D, D-1, \ldots, 1) \cdot t, \quad t \in \mathbb{R}, \\ \sum_{i=1}^{D} (D+1-i)\,|N_i| = (D+1)(n'-1) - \mathrm{d}n'. \end{cases}$$

Solving this set of equations gives:

$$t = \frac{(D+1)(n'-1) - \mathrm{d}n'}{\sum_{i=1}^{D} i^2}.$$

It can easily be verified that for $d > 2$ stated in the precondition of the proposition, the located point satisfies the inequality $\sum_{i=1}^{D} |N_i| \leq n' - 1$ imposed by the semantics of the sets $N_1, N_2, \ldots, N_D$.

Thus, the square distance between $(0, 0, \ldots, 0)$ and any point in the half-space (35) is:

$$\sum_{i=1}^{D} |N_i|^2 \geq \sum_{i=1}^{D} (i \cdot t)^2 = \frac{((D+1)(n'-1) - \mathrm{d}n')^2}{\sum_{i=1}^{D} i^2}.$$

Using the well-known formula

$$\sum_{i=1}^{D} i^2 = \frac{D(D+1)(2D+1)}{6},$$

we finally get:

$$\sum_{i=1}^{D} |N_i|^2 \geq \frac{6((D+1)(n'-1) - \mathrm{d}n')^2}{D(D+1)(2D+1)}.$$

Combining the latter with (33) finishes the proof. □

Appendix D: Proof of Proposition 8

Proof (of Proposition 8) For $a = b$, $I(a) = \emptyset$ or $I(b) = \emptyset$, the same reasoning as for Proposition 1 applies. We thus further consider $a \neq b$, $I(a) \neq \emptyset$ and $I(b) \neq \emptyset$ and prove the proposition by induction over the iteration number k.

Induction Basis For $k = 0$, the estimate obviously holds, as $R_0(a, b) - R_0^{\delta_0}(a, b) = 0$.

Inductive Step During the inductive step, we refer to Δ as $\Delta(k)$ when stressing that summation is performed from 1 to k, and as $\Delta(k+1)$ when stressing that summation is performed from 1 to $(k+1)$.

Provided that the proposition holds for k, let us estimate the difference $R_{k+1}(a, b) - R_{k+1}^{\delta_{k+1}}(a, b)$ for $(k+1)$.

Two possible cases will be considered separately: the one for $R_{k+1}^{\delta_{k+1}}(a, b) = 0$ and the other for $R_{k+1}^{\delta_{k+1}}(a, b) \neq 0$:

1. If $R_{k+1}^{\delta_{k+1}}(a,b) = 0$, then it follows from (19) and (20) that

$$\frac{C}{|I(a)|\,|I(b)|}\sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|} R_k^{\delta_k}(I_i(a), I_j(b)) \le \delta_{k+1} \tag{36}$$

and the difference $R_{k+1}(a,b) - R_{k+1}^{\delta_{k+1}}(a,b)$ is estimated thus:

$$\begin{aligned}
&R_{k+1}(a,b) - R_{k+1}^{\delta_{k+1}}(a,b) = R_{k+1}(a,b)\\
&\le \{\text{using (36)}\} \le R_{k+1}(a,b) + \delta_{k+1}\\
&\quad - \frac{C}{|I(a)|\,|I(b)|}\sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|} R_k^{\delta_k}(I_i(a), I_j(b))\\
&= \delta_{k+1} + \frac{C}{|I(a)|\,|I(b)|}\\
&\quad \times \sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|} \underbrace{\{R_k(I_i(a), I_j(b)) - R_k^{\delta_k}(I_i(a), I_j(b))\}}_{\le \Delta(k) \text{ by inductive hypothesis}}\\
&\le \delta_{k+1} + C\sum_{m=1}^{k} C^{k-m}\delta_m = \Delta(k+1).
\end{aligned}$$

2. Otherwise $R_{k+1}^{\delta_{k+1}}(a,b) \neq 0$, and thus it is defined by (19) and consequently

$$\begin{aligned}
&R_{k+1}(a,b) - R_{k+1}^{\delta_{k+1}}(a,b)\\
&= \frac{C}{|I(a)|\,|I(b)|}\\
&\quad \times \sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|} \underbrace{\{R_k(I_i(a), I_j(b)) - R_k^{\delta_k}(I_i(a), I_j(b))\}}_{\le \Delta(k)}\\
&\le C\sum_{m=1}^{k} C^{k-m}\delta_m \le \Delta(k+1).
\end{aligned}$$

The latter finishes the induction. □

References

1. Abelson, H., Sussman, G.J.: Structure and Interpretation of Computer Programs, 2nd edn. The MIT Press (1996). http://mitpress.mit.edu/sicp/full-text/book/book.html
2. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using PageRank vectors. In: FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 475–486. IEEE Computer Society, Washington, DC, USA (2006). doi:10.1109/FOCS.2006.44
3. Antonellis, I., Molina, H.G., Chang, C.C.: Simrank++: query rewriting through link analysis of the click graph. Proc. VLDB Endow. **1**(1), 408–421 (2008). doi:10.1145/1453856.1453903
4. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Comput. Networks ISDN Syst. **30**(1–7), 107–117 (1998). http://www.citeseer.ist.psu.edu/brin98anatomy.html
5. Cohen, R., Havlin, S.: Scale-free networks are ultrasmall. Phys. Rev. Lett. **90**(5), 058,701 (2003). doi:10.1103/PhysRevLett.90.058701
6. Flake, G.W., Lawrence, S., Giles, C.L.: Efficient identification of web communities. In: Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 150–160. ACM Press, New York (2000)
7. Fogaras, D., Rácz, B.: Scaling link-based similarity search. In: WWW '05: Proceedings of the 14th International Conference on World Wide Web, pp. 641–650. ACM, New York, NY, USA (2005). doi:10.1145/1060745.1060839
8. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In: Proceedings of the Twentieth International Joint Conference for Artificial Intelligence, pp. 1606–1611. Hyderabad, India (2007). http://www.cs.technion.ac.il/~shaulm/papers/pdf/Gabrilovich-Markovitch-ijcai2007.pdf
9. Ganesan, P., Garcia-Molina, H., Widom, J.: Exploiting hierarchical domain structure to compute similarity. ACM Trans. Inf. Syst. **21**(1), 64–93 (2003). doi:10.1145/635484.635487
10. Geerts, F., Mannila, H., Terzi, E.: Relational link-based ranking. In: VLDB'2004: Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp. 552–563. VLDB Endowment (2004)
11. Gleich, D.: Fast parallel pagerank: a linear system approach. Technical report (2004)
12. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: KDD '02: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 538–543. ACM Press, New York (2002). doi:10.1145/775047.775126
13. Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Exploiting the block structure of the web for computing pagerank. Technical report (2003)
14. Kohlschütter, C., Chirita, P.A., Chirita, R., Nejdl, W.: Efficient parallel computation of pagerank. In: In Proceedings of the 28th European Conference on Information Retrieval, pp. 241–252 (2006)
15. Kronrod, M., Arlazarov, V., Dinic, E., Faradzev, I.: On economic construction of the transitive closure of a direct graph. Sov. Math (Doklady) **11**, 1209–1210 (1970)
16. Li, L., Alderson, D., Tanaka, R., Doyle, J.C., Willinger, W.: Towards a theory of scale-free graphs: definition, properties, and implications (extended version). CoRR abs/cond-mat/0501169 (2005)
17. Liberty, E., Zucker, S.W.: The mailman algorithm: a note on matrix-vector multiplication. Inf. Process. Lett. **109**(3), 179–182 (2009). http://www.cs.yale.edu/homes/el327/papers/mailmanAlgorithm.pdf
18. Lin, D.: An information-theoretic definition of similarity. In: Proceedings of 15th International Conference on Machine Learning, pp. 296–304. Morgan Kaufmann, San Francisco, CA (1998). citeseer.ist.psu.edu/95071.html
19. Lin, Z., King, I., Lyu, M.R.: PageSim: a novel link-based similarity measure for the world wide web. In: WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, pp. 687–693. IEEE Computer Society, Washington, DC, USA (2006). doi:10.1109/WI.2006.127
20. Lizorkin, D., Medelyan, O., Grineva, M.: Analysis of community structure in wikipedia. In: WWW '09: Proceedings of the 18th International Conference on World Wide Web, pp. 1221–1222. ACM, New York, NY, USA (2009). doi:10.1145/1526709.1526938
21. Lizorkin, D., Velikhov, P., Grinev, M., Turdakov, D.: Accuracy estimate and optimization techniques for SimRank computation. PVLDB **1**(1), 422–433 (2008)
22. Lu, W., Janssen, J., Milios, E.E., Japkowicz, N.: Node similarity in networked information spaces. In: Stewart, D.A.,

Johnson, J.H. (eds.) CASCON, p. 11. IBM (2001). http://dblp.uni-trier.de/db/conf/cascon/cascon2001.html#LuJMJ01
23. Maguitman, A.G., Menczer, F., Erdinc, F., Roinestad, H., Vespignani, A.: Algorithmic computation and approximation of semantic similarity. World Wide Web **9**(4), 431–456 (2006). http://portal.acm.org/citation.cfm?id=1210403.1210410
24. Manaskasemsak, B., Rungsawang, A.: Parallel pagerank computation on a gigabit PC cluster. In: AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications, vol. 1, pp. 273-277. IEEE Computer Society, Washington, DC, USA (2004)
25. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **69**(2), 1–15 (2004). http://www.ncbi.nlm.nih.gov/pubmed/14995526
26. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999). http://ilpubs.stanford.edu:8090/422/
27. Shi, S., Yu, J., Yang, G., Wang, D.: Distributed page ranking in structured p2p networks. In: In ICPP, pp. 179–186 (2003)
28. Small, H.: Co-citation in the scientific literature: a new measure of the relationship between two documents. J. Am. Soc. Inf. Sci. **24**(4), 265–269 (1973)
29. Song, C., Havlin, S., Makse, H.A.: Self-similarity of complex networks (2005). http://arxiv.org/abs/cond-mat/0503078
30. Strube, M., Ponzetto, S.: WikiRelate! Computing semantic relatedness using Wikipedia. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), pp. 1419–1424. Boston, Mass. (2006)
31. Xi, W., Fox, E.A., Fan, W., Zhang, B., Chen, Z., Yan, J., Zhuang, D.: SimFusion: measuring similarity using unified relationship matrix. In: SIGIR '05: Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 130–137. ACM, New York, NY, USA (2005)
32. Zesch, T., Gurevych, I.: Analysis of the Wikipedia category graph for NLP applications. In: Proceedings of the TextGraphs-2 Workshop (NAACL-HLT 2007), pp. 1–8 (2007)